Numerical Methods for Ordinary Differential Equations: Initial Value Problems

LIM, Roktaek

Ulsan National Institute of Science and Technology

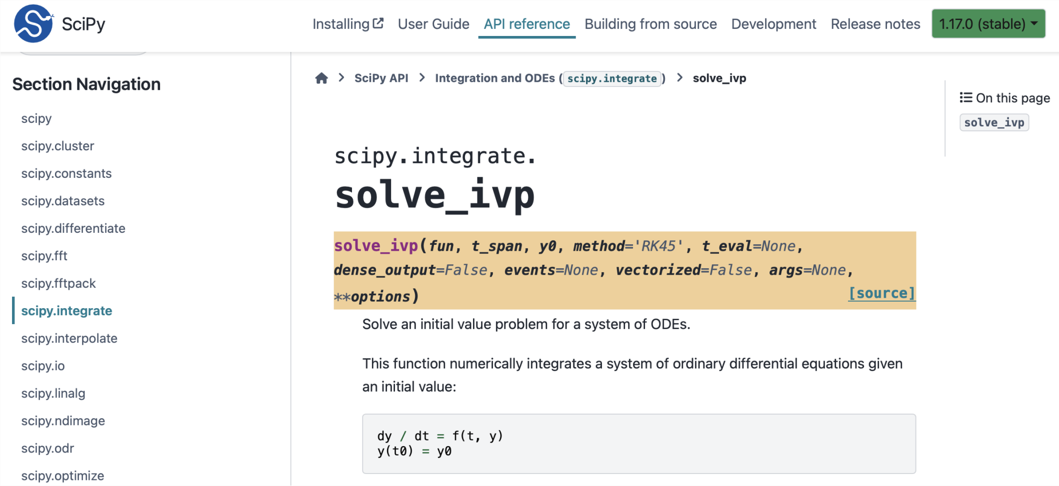
2026 Winter School on Numerical Relativity and Gravitational Waves

Learning Objectives

What You Will Learn Today

1. Basic concepts of ordinary differential equations (ODEs) and initial value problems (IVPs)
   - Brief review of ODEs and IVPs
   - Motivation for numerical methods to IVPs
2. Introduction to Runge–Kutta methods
   - Principles and formulation of Runge–Kutta methods
   - Accuracy and stability of Runge–Kutta methods

Why Learn Runge–Kutta Methods?



SciPy

Installing User Guide API reference Building from source Development Release notes 1.17.0 (stable)

Section Navigation

- scipy
- scipy.cluster
- scipy.constants
- scipy.datasets
- scipy.differentiate
- scipy.fft
- scipy.fftpack
- **scipy.integrate**
- scipy.interpolate
- scipy.io
- scipy.linalg
- scipy.ndimage
- scipy.odr
- scipy.optimize

SciPy API > Integration and ODEs (`scipy.integrate`) > solve_ivp

On this page: `solve_ivp`

scipy.integrate.
solve_ivp

`solve_ivp(fun, t_span, y0, method='RK45', t_eval=None, dense_output=False, events=None, vectorized=False, args=None, **options)` [source]

Solve an initial value problem for a system of ODEs.

This function numerically integrates a system of ordinary differential equations given an initial value:

```
dy / dt = f(t, y)
y(t0) = y0
```

Why Learn Runge–Kutta Methods?

SciPy

Installing | User Guide | API reference | Building from source | Development | Release notes | 1.17.0 (stable)

Section Navigation

- scipy
- scipy.cluster
- scipy.constants
- scipy.datasets
- scipy.differentiate
- scipy.fft
- scipy.fftpack
- **scipy.integrate**
- scipy.interpolate
- scipy.io
- scipy.linalg
- scipy.ndimage
- scipy.odr
- scipy.optimize
- scipy.signal
- scipy.sparse
- scipy.spatial
- scipy.special
- scipy.stats

`solve_ivp`(fun, t_span, y0[, method, t_eval, ...])	Solve an initial value problem for a system of ODEs.
`RK23`(fun, t0, y0, t_bound[, max_step, rtol, ...])	Explicit Runge-Kutta method of order 3(2).
`RK45`(fun, t0, y0, t_bound[, max_step, rtol, ...])	Explicit Runge-Kutta method of order 5(4).
`DOP853`(fun, t0, y0, t_bound[, max_step, ...])	Explicit Runge-Kutta method of order 8.
`Radau`(fun, t0, y0, t_bound[, max_step, ...])	Implicit Runge-Kutta method of Radau IIA family of order 5.
`BDF`(fun, t0, y0, t_bound[, max_step, rtol, ...])	Implicit method based on backward-differentiation formulas.
`LSODA`(fun, t0, y0, t_bound[, first_step, ...])	Adams/BDF method with automatic stiffness detection and switching.
`OdeSolver`(fun, t0, y0, t_bound, vectorized)	Base class for ODE solvers.
`DenseOutput`(t_old, t)	Base class for local interpolant over step made by an ODE solver.
`OdeSolution`(ts, interpolants[, alt_segment])	Continuous ODE solution.

Back to top

On this page

- Integrating functi function object
- Integrating functi fixed samples
- Summation
- **Solving initial va for ODE systems**
  - Old API
- Solving boundary problems for ODI

Why Learn Runge–Kutta Methods?

MATLAB 도움말 센터 커뮤니티 학습 MATLAB 받기 로그인

목차

« 문서 홈

« MATLAB

« 수학

« 수치 적분과 미분 방정식

« 상미분 방정식

ODE 솔버 선택하기

이 페이지 내용

상미분 방정식

ODE의 유형

연립 ODE

고계 ODE(Higher-Order ODE)

복소 ODE(Complex ODE)

기본적인 솔버 선택

ODE 예제와 파일에 대한 요약

참고 문헌

참고 항목

문서 예제 함수 앱 비디오 Answers

기본적인 솔버 선택

ode45는 대부분의 ODE 문제에서 잘 동작하며, 일반적으로 첫 번째로 선택하는 솔버가 됩니다. 그러나, 정확도 요구 사항이 더 느슨하거나 더 엄격한 문제의 경우 ode23, ode78, ode89 및 ode113이 ode45보다 더 효율적일 수 있습니다.

일부 ODE 문제는 *경직성(Stiff)*을 보이거나 계산하기 어렵습니다. 경직성은 정확히 정의하기 곤란한 용어지만, 일반적으로 문제의 어느 부분에서 스케일링에 차이가 있는 경우 발생합니다. 예를 들어, 2개의 해 성분이 서로 크게 다른 시간 스케일에서 변하고 있는 ODE는 경직성 방정식일 수 있습니다. 비경직성 솔버(예: ode45)가 문제를 풀 수 없거나 속도가 매우 느린 경우 이 문제를 경직성 문제로 여길 수 있습니다. 비경직성 솔버의 속도가 매우 느린 경우에는 ode15s 등의 경직성 솔버를 대신 사용해 보십시오. 경직성 솔버를 사용하면 야코비 행렬(Jacobian Matrix)이나 그 희소성 패턴을 제공하여 안정성과 효율성을 향상시킬 수 있습니다.

ode 객체를 사용하여 문제의 속성에 따라 솔버 선택을 자동화할 수 있습니다. 어떤 솔버를 사용해야 할지 잘 모르는 경우 다음 표를 참조하십시오. 이 표에는 각 솔버를 언제 사용할 수 있는지에 대한 일반적인 지침이 나와 있습니다.

솔버	문제 유형	정확도	사용하는 경우
ode45	비경직성(Nonstiff)	중간	대부분의 경우 사용할 수 있습니다. 다른 솔버보다 ode45를 가장 먼저 시도해 봐야 합니다.
ode23		낮음	허용오차가 엄격하지 않거나 반경직성이 있는 문제의 경우 ode23이 ode45보다 더 효율적일 수 있습니다.
ode113		낮음 ~ 높음	엄격한 허용오차를 가지는 문제에서나 ODE 함수를 계산하는 데 시간이 많이 걸리는 경우 ode113이 ode45보다 더 효율적일 수 있습니다.
ode78		높음	정확도 요구 사항이 높은 매끄러운 해를 갖는 문제에서는 ode78이 ode45보다 더 효율적일 수 있습니다.
ode89		높음	긴 시간 구간에 대해 적분을 수행하거나 허용오차를 특히 엄격하게 할 때는 매우 매끄러운 문제에서 ode89가 ode78보다 더 효율적일 수 있습니다.

Why Learn Runge–Kutta Methods?

HOME | MODELING | SOLVERS | ANALYSIS | MACHINE LEARNING | DEVELOPER TOOLS | COMMERCIAL SUPPORT

Search everywhere... /

DifferentialEquations.jl

Search docs (Ctrl + /)

ODE Solvers

- Recommended Methods
- Translations from MATLAB/Python/R
- Full List of Methods

Non-autonomous Linear ODE / Lie Group ODE Solvers

Dynamical, Hamiltonian, and 2nd Order ODE Solvers

Split ODE Solvers

Steady State Solvers

BVP Solvers

SDE Solvers

SDAE Solvers

Version v7.20.0

Explicit Runge-Kutta Methods

- `Euler` - The canonical forward Euler method. Fixed timestep only.
- `Midpoint` - The second order midpoint method. Uses embedded Euler method for adaptivity.
- `Heun` - The second order Heun's method. Uses embedded Euler method for adaptivity.
- `Ralston` - The optimized second order midpoint method. Uses embedded Euler method for adaptivity.
- `RK4` - The canonical Runge-Kutta Order 4 method. Uses a defect control for adaptive stepping using maximum error over the whole interval.
- `BS3` - Bogacki-Shampine 3/2 method.
- `OwrenZen3` - Owren-Zennaro optimized interpolation 3/2 method (free 3rd order interpolant).
- `OwrenZen4` - Owren-Zennaro optimized interpolation 4/3 method (free 4th order interpolant).
- `OwrenZen5` - Owren-Zennaro optimized interpolation 5/4 method (free 5th order interpolant).
- `DP5` - Dormand-Prince's 5/4 Runge-Kutta method. (free 4th order interpolant).
- `Tsit5` - Tsitouras 5/4 Runge-Kutta method. (free 4th order interpolant).
- `Anas5(w)` - 4th order Runge-Kutta method designed for periodic problems. Requires a periodicity estimate `w` which when accurate the method becomes 5th order (and is otherwise 4th order with less error for better estimates).
- `FRK65(w=0)` - Zero Dissipation Runge-Kutta of 6th order. Takes an optional argument `w` to for the periodicity phase, in which case this method results in zero numerical dissipation.
- `PFRK87(w=0)` - Phase-fitted Runge-Kutta of 8th order. Takes an optional argument `w` to for the periodicity phase, in which case this method results in zero numerical dissipation.
- `RKO65` - Tsitouras' Runge-Kutta-Oliver 6 stage 5th order method. This method is robust on problems which have a singularity at `t=0`.
- `TanYam7` - Tanaka-Yamashita 7 Runge-Kutta method.
- `DP8` - Hairer's 8/5/3 adaption of the Dormand-Prince Runge-Kutta method. (7th order interpolant).
- `TsitPap8` - Tsitouras-Papakostas 8/7 Runge-Kutta method.
- `Feagin10` - Feagin's 10th-order Runge-Kutta method.

Ordinary Differential Equations

ODEs as Mathematical Models

Mathematical models describe how quantities change.

- The Logistic Model

$$\frac{\mathrm{d}}{\mathrm{d}t} y(t) = r y(t) \left[1 - \frac{y(t)}{K} \right].$$

- Mass–Spring System

$$m \frac{\mathrm{d}^2}{\mathrm{d}t^2} y(t) + k y(t) = 0.$$

- Newton's Law of Cooling

$$\frac{\mathrm{d}}{\mathrm{d}t} y(t) = -k \left[y(t) - y_{\mathrm{env}} \right].$$

- ...

Second-Order ODE: Two body problem

Two body problem

Let $\mathbf{x}_1(t)$ and $\mathbf{x}_2(t)$ be the positions of two bodies, and m_1 and m_2 be their masses. Let

$$\begin{aligned}\mathbf{x}(t) &= \mathbf{x}_2(t) - \mathbf{x}_1(t),\\ \|\mathbf{x}\| &= \sqrt{\mathbf{x}\cdot\mathbf{x}}.\end{aligned}$$

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}\mathbf{x}(t) = -\frac{G\,(m_1+m_2)}{\|\mathbf{x}(t)\|^3}\mathbf{x}(t),$$

where G is the gravitational constant.

$\mathbf{x}$ is called dependent variable and t is an independent variable. The two body problem is described by a second-order ODE.

System of ODEs: The Lotka–Volterra predator–prey model

The Lotka–Volterra predator–prey model

$$\frac{\mathrm{d}}{\mathrm{d}t}u(t) = \alpha u(t) - \beta u(t)v(t),$$
$$\frac{\mathrm{d}}{\mathrm{d}t}v(t) = -\gamma v(t) + \delta u(t)v(t),$$

where u is the population density of the prey, v is the population density of the predator; α, β, γ, and δ are model parameters.

u and v are called dependent variables, and t is an independent variable. The Lotka–Volterra model consists of two coupled, first-order ODEs.

System of ODEs: The Lotka–Volterra predator–prey model

The Lotka–Volterra predator–prey model

Let

$$\mathbf{x}(t) = \begin{bmatrix} u(t) \\ v(t) \end{bmatrix}, \; \mathbf{f}(t, \mathbf{x}(t)) = \begin{bmatrix} \alpha u(t) - \beta u(t) v(t) \\ -\gamma v(t) + \delta u(t) v(t) \end{bmatrix}$$

Then,

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) = \mathbf{f}(t, \mathbf{x}(t))$$

Higher-Order ODEs

An ODE of order n is an equation of the form:

$$\frac{\mathrm{d}^n}{\mathrm{d}t^n} y(t) = f\left(t, y(t), \frac{\mathrm{d}}{\mathrm{d}t} y(t), \ldots, \frac{\mathrm{d}^{n-1}}{\mathrm{d}t^{n-1}} y(t)\right).$$

We can reduce the ODE of order n into an ODE of order 1. Let

$$\mathbf{x}(t) = \begin{bmatrix} y(t) \\ \frac{\mathrm{d}}{\mathrm{d}t} y(t) \\ \vdots \\ \frac{\mathrm{d}^{n-1}}{\mathrm{d}t^{n-1}} y(t) \end{bmatrix}, \quad \mathbf{f}(t, \mathbf{x}(t)) = \begin{bmatrix} \frac{\mathrm{d}}{\mathrm{d}t} y(t) \\ \frac{\mathrm{d}^2}{\mathrm{d}t^2} y(t) \\ \vdots \\ f\left(t, y(t), \frac{\mathrm{d}}{\mathrm{d}t} y(t), \ldots, \frac{\mathrm{d}^{n-1}}{\mathrm{d}t^{n-1}} y(t)\right) \end{bmatrix}.$$

Then, the ODE of order n is equivalent to the following:

$$\frac{\mathrm{d}}{\mathrm{d}t} \mathbf{x}(t) = \mathbf{f}(t, \mathbf{x}(t)).$$

Higher-Order ODEs

Consider the second-order ODE given by

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}y(t) + p(t)\frac{\mathrm{d}}{\mathrm{d}t}y(t) + q(t)y(t) = g(t).$$

Then, we have

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} y(t) \\ \frac{\mathrm{d}}{\mathrm{d}t}y(t) \end{bmatrix} &= \begin{bmatrix} \frac{\mathrm{d}}{\mathrm{d}t}y(t) \\ -p(t)\frac{\mathrm{d}}{\mathrm{d}t}y(t) - q(t)y(t) + g(t) \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 \\ -q(t) & -p(t) \end{bmatrix}\begin{bmatrix} y(t) \\ \frac{\mathrm{d}}{\mathrm{d}t}y(t) \end{bmatrix} + \begin{bmatrix} 0 \\ g(t) \end{bmatrix}.\end{aligned}$$

Initial conditions

Solutions to ODEs are usually not unique due to the appearance of integration constants.

A simple second-order ODE

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}y(t) = a.$$

Which leads to

$$\frac{\mathrm{d}}{\mathrm{d}t}y(t) = at + C_0,\ \ y(t) = \frac{1}{2}at^2 + C_0 t + C_1.$$

This contains two integration constants. Standard practice would be to specify $\frac{\mathrm{d}}{\mathrm{d}t}y(0) = C_0$ and $y(0) = C_1$. These are initial conditions.

Initial Value Problems

The first-order differential equation for the function $y(t)$ is written as

$$\frac{\mathrm{d}}{\mathrm{d}t} y(t) = f(t, y(t)), \tag{1}$$

where $f(t, y(t))$ can be any function of the independent variable t and the dependent variable y.

The differential equation will be considered with an initial condition:

$$y(t_0) = y_0. \tag{2}$$

The differential (1) together with the initial condition (2) is called an initial value problem.

In general, an initial value problem takes the form

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{y}(t) = \mathbf{f}(t, \mathbf{y}(t)), \\ \mathbf{y}(t_0) = \mathbf{y}_0, \end{cases}$$

where

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{y}(t) = \frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_n(t) \end{pmatrix} = \begin{pmatrix} f_1(t, \mathbf{y}(t)) \\ f_2(t, \mathbf{y}(t)) \\ \vdots \\ f_n(t, \mathbf{y}(t)) \end{pmatrix}.$$

Motivation for Numerical Methods in Ordinary Differential Equations

Motivation for Numerical Methods in ODEs

$$E + S \underset{k_2}{\overset{k_1}{\rightleftharpoons}} ES \xrightarrow{k_3} E + P$$

Enzyme–Substrate Reaction Models

$$\frac{\mathrm{d}}{\mathrm{d}t}S(t) = -k_1 E(t)S(t) + k_2 C(t), \qquad \frac{\mathrm{d}}{\mathrm{d}t}E(t) = -k_1 E(t)S(t) + (k_2 + k_3)C(t),$$

$$\frac{\mathrm{d}}{\mathrm{d}t}C(t) = k_1 E(t)S(t) - (k_2 + k_3)C(t), \qquad \frac{\mathrm{d}}{\mathrm{d}t}P(t) = k_3 C(t),$$

where $S(t)$, $E(t)$, $C(t)$, and $P(t)$ denote the concentrations of substrate, free enzyme, enzyme–substrate complex, and product, respectively.

The solutions of the model cannot be expressed in closed form.

Method of integrating factors

A first-order linear ODE has the form:

$$\frac{\mathrm{d}}{\mathrm{d}t}y(t) + P(t)y(t) = Q(t).$$

The integrating factor is given by

$$\mu(t) = e^{\int P(t)\,\mathrm{d}t}.$$

Then, the solution can be written as

$$y(t) = \frac{1}{\mu(t)}\left[\int \mu(t)Q(t)\,\mathrm{d}t + C\right].$$

Motivation for Numerical Methods in ODEs

Method of integrating factors

$$\mu(t) = e^{\int P(t)\, \mathrm{d}t}, \; y(t) = \frac{1}{\mu(t)} \left[\int \mu(t) Q(t)\, \mathrm{d}t + C \right].$$

Most ODEs do not have solutions expressible in closed form. For example, the following integral

$$\int_a^b e^{-t^2}\, \mathrm{d}t$$

cannot be evaluated in closed form.

We must rely on numerical methods that produce approximations to the desired solutions.

Numerical Methods for Initial Value Problems

Purpose of Numerical Methods for IVPs

An IVP for a first-order ODE is given by

$$\frac{\mathrm{d}}{\mathrm{d}t}y(t) = f(t, y(t)),\ y(t_0) = y_0.$$

- $y(t_j)$: the exact solution to the problem at $t_j > t_0$,
- y_j: the approximate solution at t_j.

The goal is to compute an approximate solution

$$y_j \approx y(t_j).$$

Initial Value Problem

$$\frac{\mathrm{d}}{\mathrm{d}t}y(t) = f(t, y(t)), \; y(t_0) = y_0.$$

The equation of the tangent line of $y(t)$ at $t = t_0$ is expressed by

$$\hat{y}(t) = y_0 + f(t_0, y_0)(t - t_0). \qquad (3)$$

If $t_1 = t_0 + h$ is close to t_0, we can approximate $y(t_1)$ using (3).

$$y(t_1) \approx y_1 = y_0 + hf(t_0, y_0).$$

This is the Euler method for solving IVPs.

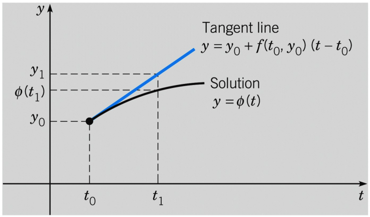


Figure 2.7.2

The Fundamental Theorem of Calculus

Initial Value Problem

$$\frac{\mathrm{d}}{\mathrm{d}t}y(t) = f(t, y(t)),\ y(t_0) = y_0.$$

By integrating $f(t, y(t))$ from t_0 to $t_0 + h$ and applying the fundamental theorem of calculus, we obtain:

$$\int_{t_0}^{t_0+h} f(t, y(t))\,\mathrm{d}t = y(t_0 + h) - y(t_0).$$

The integral form of IVP

$$y(t_0 + h) = y(t_0) + \int_{t_0}^{t_0+h} f(t, y(t))\,\mathrm{d}t \tag{4}$$

By approximating the integral in (4) using numerical quadrature rules, we can obtain

$$y(t_1) \approx y_1 = y_0 + hK(h, t_0, y_0)$$

where

$$t_1 = t_0 + h,\ \ hK(h, t_0, y_0) \approx \int_{t_0}^{t_0+h} f(t, y(t))\,\mathrm{d}t.$$

The key point is that $K(h, t_0, y_0)$ depend on $y_0 = y(t_0)$, but does not depend on $y(t)$ for any $t \neq t_0$.

Examples of Numerical Quadrature Rules

- Midpoint rule:

$$\int_a^b f(x)\,\mathrm{d}x \approx (b-a)f\left(\frac{a+b}{2}\right).$$

- Trapezoidal rule:

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{(b-a)}{2}\left[f(a)+f(b)\right].$$

- Simpson's rule:

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{(b-a)}{6}\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right].$$

Midpoint Rule

The midpoint rule approximates the integral in (4) as

$$\int_{t_0}^{t_0+h} f(t, y(t))\, \mathrm{d}t \approx hf\left(t_0 + \frac{h}{2}, y\left(t_0 + \frac{h}{2}\right)\right).$$

If we approximate $y\left(t_0 + \frac{h}{2}\right)$ using the Euler method, we obtain

$$y_1 = y_0 + hf\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2} f(t_0, y_0)\right).$$

Runge's Method

Statt

(1) $$\Delta y = f(x_0 y_0)\,\Delta x \quad \text{u. s. w.}$$

ist es schon viel besser wenn man

(2) $$\Delta y = f\left(x_0 + \tfrac{1}{2}\Delta x,\; y_0 + \tfrac{1}{2} f(x_0 y_0)\,\Delta x\right)\Delta x$$

u. s. w.

Runge, C. Ueber die numerische Auflösung von Differentialgleichungen. Math. Ann. 46, 167–178 (1895).
`https://doi.org/10.1007/BF01446807`

The midpoint method

$$y_1 = y_0 + hf\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2} f(t_0, y_0)\right).$$

In practice,

$$k_1 = f(t_0, y_0),$$
$$k_2 = f\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2} k_1\right),$$
$$y_1 = y_0 + hk_2.$$

Heun's Method

The trapezoidal rule approximates the integral in (4) as

$$\int_{t_0}^{t_0+h} f(t, y(t))\ \mathrm{d}t \approx \frac{h}{2}\left[f(t_0, y(t_0)) + f(t_0 + h, y(t_0 + h))\right].$$

If we approximate $y(t_0 + h)$ using the Euler method, we obtain

$$\begin{aligned} k_1 &= f(t_0, y_0), \\ k_2 &= f(t_0 + h, y_0 + hk_1), \\ y_1 &= y_0 + \frac{h}{2}\left(k_1 + k_2\right). \end{aligned}$$

This method is also called the improved Euler method.

An explicit one-step method is a method which, given y_0 at t_0 computes a sequence of approximations $y_1, \ldots, y_N$ to the solution of an IVP at time steps $t_1, \ldots, t_N$ using an update formula of the form:

$$y(t_n) \approx y_n = y_{n-1} + h_n K(h_n, t_{n-1}, y_{n-1}), \; h_n = t_n - t_{n-1}$$

for $n = 1, \ldots, N$.
The method is called one-step method because the value y_n explicitly depends only on the value y_{n-1} and $f(t_{n-1}, y_{n-1})$.

The Explicit s-stage Runge–Kutta Methods

The explicit s-stage Runge–Kutta methods are one-step methods that uses s evaluations of $f(t, y(t))$ with the representation

$$k_i = f\left(t_0 + c_i h, y_0 + h\sum_{j=1}^{i-1} a_{ij}k_j\right),\ i = 1, \ldots, s,$$

$$y_1 = y_0 + h\sum_{i=1}^{s} b_i k_i.$$

Note that $c_1 = 0$.

Example: Two-Stage Explicit Runge–Kutta Methods

1. The midpoint method

$$
\begin{aligned}
k_1 &= f(t_0, y_0), \\
k_2 &= f\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2} k_1\right), \\
y_1 &= y_0 + h k_2.
\end{aligned}
$$

2. Heun's method

$$
\begin{aligned}
k_1 &= f(t_0, y_0), \\
k_2 &= f(t_0 + h, y_0 + h k_1), \\
y_1 &= y_0 + h\left(\frac{1}{2} k_1 + \frac{1}{2} k_2\right).
\end{aligned}
$$

The Global Error

Let $y(t)$ be a solution of the IVP

$$\frac{\mathrm{d}}{\mathrm{d}t} y(t) = f(t, y(t)),\ y(t_0) = y_0$$

on the interval $[t0, t_N]$.

The global error of the one-step method

$$e_N = |y(t_N) - y_N|.$$

It is the difference between the solution $y(t_N)$ of the IVP at t_N and the result of the one-step method at t_N.

The Local Error

Let $y(t)$ be a solution of the IVP

$$\frac{\mathrm{d}}{\mathrm{d}t} y(t) = f(t, y(t)), \; y(t_0) = y_0$$

on the interval $[t_{n-1}, t_n]$.

The local error of the one-step method

$$l_n = y(t_n) - \left[y(t_{n-1}) + h_n K(h_n, t_{n-1}, y(t_{n-1})) \right].$$

It is the difference between $y(t_n)$ and the result of the one-step method with the exact initial value $y(t_{n-1})$.

The Local Error of the Euler Method

We will use the Taylor expansion of $y(t)$ at t_0. Let

$$f_t = \frac{\partial}{\partial t} f(t_0, y_0),\ f_y = \frac{\partial}{\partial y} f(t_0, y_0),\ f_0 = f(t_0, y_0).$$

And we will use the following:

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2} y(t_0) = f_t + f_y \frac{\mathrm{d}}{\mathrm{d}t} y(t_0) = f_t + f_y f_0.$$

Then,

$$\begin{aligned} y(t_1) &= y_0 + h f_0 + \frac{h^2}{2} + \cdots . \\ y_1 &= y_0 + h f_0. \\ l_1 &= \mathcal{O}(h^2). \end{aligned}$$

The Local Error of the Midpoint Method

We will use the Taylor expansion of $y(t)$ at t_0. We have

$$f(t_0 + ah, y_0 + bhf(t_0, y(t_0))) = f_0 + ahf_t + bhf_y f_0 + \frac{a^2h^2}{2} f_{tt} + abh^2 f_{ty} f_0 \\ + \frac{1}{2} b^2 h^2 f_{yy} f_0^2 + \cdots$$

Then,

$$\begin{aligned} y(t_1) &= y_0 + hf_0 + \frac{h^2}{2}(f_t + f_y f_0) + \frac{h^3}{6}\left(f_{tt} + 2f_{ty}f_0 + f_{yy}f^2 + f_y f_t + f_y^2 f_0\right) \\ &\quad + \cdots . \\ y_1 &= y_0 + hf\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2} f(t_0, y_0)\right) \\ &= y_0 + hf_0 + \frac{h^2}{2}(f_t + f_y f_0) + \frac{h^3}{8}\left(f_{tt} + 2f_{ty}f_0 + f_{yy}f_0^2\right) + \cdots . \\ l_1 &= \mathcal{O}(h^3). \end{aligned}$$

The Truncation Error

The local error on the interval $[t_{n-1}, t_n]$ is given by

$$l_n = y(t_n) - \left[y(t_{n-1}) + h_n K(h_n, t_{n-1}, y(t_{n-1}))\right].$$

The truncation error

The truncation error is the quotient of the local error and h_n, defined as:

$$\tau_n = \frac{l_n}{h_n} = \frac{y(t_n) - y(t_{n-1})}{h_n} - K(h_n, t_{n-1}, y(t_{n-1})).$$

Order of Accuracy of the One-Step Method

The truncation error is on the interval $[t_{n-1}, t_n]$ is given by

$$\tau_n = \frac{l_n}{h_n} = \frac{y(t_n) - y(t_{n-1})}{h_n} - K(h_n, t_{n-1}, y(t_{n-1})).$$

The one-step method is consistent and has order of accuracy p, if there exists a constant D independent of $h = \max_{n=1,\ldots,N} h_n$ such that

$$\max_{n=1,\ldots,N} |\tau_n| \leq Dh^p.$$

The Order of Accuracy of the Euler and Midpoint Methods

Let

$$h = \max_{n=1,\ldots,N} h_n$$

- Euler method: $l_n = \mathcal{O}(h_n^2)$ for $n = 1, \ldots, N$.

$$\tau_n = \mathcal{O}(h_n), \; n = 1, \ldots, N.$$

$$\max_{n=1,\ldots,N} |\tau_n| \leq D_{\text{Euler}} h$$

- Midpoint method: $l_n = \mathcal{O}(h_n^3)$ for $n = 1, \ldots, N$.

$$\tau_n = \mathcal{O}(h_n^2), \; n = 1, \ldots, N.$$

$$\max_{n=1,\ldots,N} |\tau_n| \leq D_{\text{Midpoint}} h^2$$

Note that D_{Euler} and D_{Midpoint} are independent of h.

Order conditions of Explicit Runge–Kutta Methods

The explicit s-stage Runge–Kutta methods are represented as follows:

$$k_i = f\left(t_0 + c_i h, y_0 + h\sum_{j=1}^{i-1} a_{ij}k_j\right),\ i = 1, \dots, s,$$

$$y_1 = y_0 + h\sum_{i=1}^{s} b_i k_i.$$

The order conditions derived by using the followings:

$$f = f(t, y(t)),\ \frac{\mathrm{d}^2}{\mathrm{d}t^2}y(t) = f_t + f_y\frac{\mathrm{d}}{\mathrm{d}t}y(t) = f_t + f_y f.$$

$$f(t_0 + ah, y_0 + bhf(t_0, y(t_0))) = f_0 + ahf_t + bhf_y f_0 + \frac{a^2h^2}{2}f_{tt} + abh^2 f_{ty} f_0 + \frac{1}{2}b^2h^2 f_{yy} f_0^2 + \cdots .$$

Order conditions of Explicit Runge–Kutta Methods

- Order 1

$$\sum_{i=1}^{s} b_i = 1.$$

- Order 2

$$\sum_{i=1}^{s} b_i c_i = \frac{1}{2}.$$

- Order 3

$$\sum_{i=1}^{s} b_i c_i^2 = \frac{1}{3}, \quad \sum_{i,j=1}^{s} b_i a_{ij} c_j = \frac{1}{6}.$$

- Order 4

$$\sum_{i=1}^{s} b_i c_i^3 = \frac{1}{4}, \quad \sum_{i,j=1}^{s} b_i a_{ij} c_j^2 = \frac{1}{12}, \quad \sum_{i,j,k=1}^{s} b_i a_{ij} a_{jk} c_k = \frac{1}{24}, \quad \sum_{i,j=1}^{s} b_i c_i a_{ij} c_j = \frac{1}{8}.$$

Third- and Fourth-Order Runge–Kutta Methods

Kutta's third-order method

$$
\begin{aligned}
k_1 &= f(t_0, y_0), \\
k_2 &= f\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_1\right), \\
k_3 &= f\left(t_0 + h,\ y_0 - hk_1 + 2hk_2\right), \\
y_1 &= y_0 + \frac{h}{6}\left(k_1 + 4k_2 + k_3\right).
\end{aligned}
$$

The Runge–Kutta method

$$
\begin{aligned}
k_1 &= f(t_0, y_0), \\
k_2 &= f\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_1\right), \\
k_3 &= f\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_2\right), \\
k_4 &= f(t_0 + h, y_0 + hk_3), \\
y_1 &= y_0 + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right).
\end{aligned}
$$